2

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Ada Compiler Validation Summary Report:
Control Data Corporation, CYBER 180 Ada Compiler, Version 1.1, CYBER 180-930-31 (host and target) 880624S1.09122

5. TYPE OF REPORT & PERIOD COVERED

24 June 1988-24 June 1989

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

National Bureau of Standards,
Gaithersburg, Maryland, USA

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS

National Bureau of Standards,
Gaithersburg, Maryland, USA

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Ada Joint Program Office
United States Department of Defense
Washington, DC 20301-3081

12. REPORT DATE

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office)

National Bureau of Standards,
Gaithersburg, Maryland, USA

15. SECURITY CLASS (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 If different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

CYBER 180 Ada Compiler, Version 1.1, Control Data Corporation, National Bureau of Standards, CYBER 180-930-31 under NOS/VE, Level 700 (host and target), ACVC 1.09

AVF Control Number: NBS88VCDC510

Ada Compiler
VALIDATION SUMMARY REPORT:

Certificate Number: 880624S1.09132
Control Data Corporation
CYBER 180 Ada Compiler, Version 1.1
HOST and TARGET COMPUTER:
CYBER 180-930-31

Completion of On-Site Testing:
24 June 1988

Prepared By:
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

Ada Compiler Validation Summary Report:

Compiler Name: CYBER 180 Ada Compiler, Version 1.1

Certificate Number: 880624S1.09132

Host:	Target:
CYBER 180-930-31 under NOS/VE, Level 700	CYBER 180-930-31 under NOS/VE, Level 700

Testing Completed 24 June 1988 Using ACVC 1.9

This report has been reviewed and is approved.

David K. Jefferson

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Bureau of Standards
Gaithersburg, MD 20899

John F Kramer

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria, VA 22311

William S Ritchie

Ada Joint Program Office
Mr. William S. Ritchie, Acting Director
Department of Defense
Washington DC 20301

Ada Compiler Validation Summary Report:

Compiler Name: CYBER 180 Ada Compiler, Version 1.1

Certificate Number: 880624S1.09132

Host:	Target:
CYBER 180-930-31 under NOS/VE, Level 700	CYBER 180-930-31 under NOS/VE, Level 700

Testing Completed 24 June 1988 Using ACVC 1.9

This report has been reviewed and is approved.

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Bureau of Standards
Gaithersburg, MD 20899

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria, VA 22311

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report ~~(VSR)~~ describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability ~~(ACVC)~~. An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

This information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of test are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

To attempt to identify any unsupported language constructs required by the Ada Standard

To determine that the implementation-dependent behavior is allowed by the Ada Standard

On-site testing was completed 24 June 1988 at Sunnyvale, California.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, Maryland 20899

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines. Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide., December 1986.

1.4 DEFINITION OF TERMS

ACVC The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.

Ada Commentary An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.

Ada Standard ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

Applicant The agency requesting validation.

AVF The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.

AVO The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.

Compiler A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test — An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host — The computer on which the compiler resides.

Inapplicable test — An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Language Maintenance — The Language Maintenance Panel (LMP) is a committee established by the Ada Board to recommend interpretations and Panel possible changes to the ANSI/MIL-STD for Ada.

Passed test — An ACVC test for which a compiler generates the expected result.

Target — The computer for which a compiler generates code.

Test — An Ada program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn test — An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicabilit' of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is nc: necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: CYBER 180 Ada Compiler, Version 1.1

ACVC Version: 1.9

Certificate Number: 880624S1.09132

Host Computer:

Machine: CYBER 180-930-31

Operating System: NOS/VE
Level 700

Memory Size: 64Mbytes RAM

Target Computer:

Machine: CYBER 180-930-31

Operating System: NOS/VE
Level 700

Memory Size: 64Mbytes RAM

Additional Configuration Information:

4.4 Gbytes disk drives
terminals connected using CDCNET
2 magnetic tape drives
2000 l/m printer

Communications Network: none

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See test D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined types LONG_FLOAT in the package STANDARD. (See tests B86001BC and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Expression evaluation.

  Apparently all default initialization expressions or record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

Apparently NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Apparently NUMERIC_ERROR is raised when a literal operand in a fixed point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is not gradual. (See tests C45524A..Z.)

- Rounding.

The method used for rounding to integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round away from zero. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round away from zero. (See test C4A014A.)

- Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises CONSTRAINT_ERROR. (See test C36003A.)

CONSTRAINT_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

CONSTRAINT_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises CONSTRAINT_ERROR when the array subtype is declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONTRAINT_ERROR when the array subtype is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises CONSTRAINT_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with disciminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is not supported, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are not supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are not supported. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are not supported. (See tests A39005E and C87B62C.)

Record representation clauses are not supported. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are not supported. (See test C87B62A.)

- Pragmas.

The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

- Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

There are strings which are illegal external file names for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2102C and CE2102H.)

Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D and CE2102E.)

Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I, and CE2102J.)

RESET and DELETE are supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2102G and CE2102K.)

Dynamic creation and deletion of files are supported for SEQUENTIAL_IO and DIRECT_IO. (See tests CE2106A and CE2106B.)

Overwriting to a sequential file truncates the file to last element written. (See test CE2208B.)

An existing text file can be opened in OUT_FILE mode, can not be created in OUT_FILE mode, and cannot be created in IN_FILE mode. (See test EE3102C.)

Only one internal file can be associated with each external file for text I/O for both reading and writing. (See tests CE2110B, CE2111D, CE3111A..E (5 tests), CE3114B, and CE3115A.)

Only one internal file can be associated with each external file for sequential I/O for both reading and writing. (See tests CE2107A..C (3 tests).)

Only one internal file can be associated with each external file for direct I/O for both reading and writing. (See tests CE2107E, CE2107G..I (3 tests) and CE2111H.)

More than one internal file can be associated with each external file for direct I/O for reading only. (See test CE2107F.)

An external file associated with more than one internal file cannot be deleted for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO. (See test CE2110B.)

Temporary sequential files are not given names. Temporary direct files are not given names. (See tests CE2107D..E (2 tests), CE2107H..I(2 tests) CE2108A, CE2108C, and CE3112A.)

- Generics.

Generic subprogram declarations and bodies can be compiled in separate compilations so long as no instantiations of those units precede the bodies. This compiler requires that a generic unit's body be compiled prior to instantiation, and so the unit containing the instantiations is rejected. (See test CA2009F.)

Generic package declarations and bodies can be compiled in separate compilations so long as no instantiations of those units precede the bodies. This compiler requires that a generic unit's body be compiled prior to instantiation, and so the unit containing the instantiations is rejected. (See tests CA2009C, BC3204C.)

Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

At the time of testing, version 1.9 of the ACVC comprised 3122 tests of which 27 had been withdrawn. Of the remaining tests, 112 were determined to be inapplicable to this implementation. Modifications to the code, processing, or grading for 70 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	105	1044	1757	17	14	46	2983
Inapplicable	5	7	96	0	4	0	112
Withdrawn	3	2	21	0	1	0	27
TOTAL	113	1053	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	187	558	639	242	166	98	140	326	135	36	232	3	221	2983
Inapplicable	17	14	35	6	0	0	3	1	2	0	2	0	32	112
Withdrawn	2	14	3	0	0	1	2	0	0	0	2	1	2	27
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 27 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A E28005C C34004A C35502P A35902C C35904A
C35904E C35A03E C35A03R C37213H C37213J C37215C
C37215E C37215G C37215H C38102C C41402A C45332A
C45614C A74106C C85018B C87B04B CC1311B BC3105A
AD1A01A CE2401H CE3208A

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 112 test were inapplicable for the reasons indicated:

C24113I..X (16 tests) contain contain lines whose length is greater than this implementation's maximum line length of 132 characters.

C35508I..J (2 tests) and C35508M..N (2 tests) use enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1). These clauses are not supported by this compiler.

C35702A uses SHORT_FLOAT which is not supported by this implementation.

A39005B and C87B62A use length clauses with SIZE specifications for derived integer types or for enumeration types which are not supported by this compiler.

A39005E and C87B62C use length clauses with SMALL specifications which are not supported by this implementation.

A39005G uses a record representation clause which is not supported by this compiler.

The following (14) tests use SHORT_INTEGER, which is not supported by this compiler.

C45231B C45304B C45502B C45503B C45504B
C45504E C45611B C45613B C45614B C45631B
C45632B B52004E C55B07B B55B09D

The following (13) tests use LONG_INTEGER, which is not supported by this compiler.

C45231C C45304C C45502C C45503C C45504C
C45504F C45611C C45613C C45631C C45632C
B52004D C55B07A B55B09C

C45231D requires a macro substitution for any predefined numeric types other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT, and LONG_FLOAT. This compiler does not support any such types.

C4A013B uses a static value that is within the range of the most accurate floating point base type, and MACHINE_OVERFLOWS is false for this type. The test executes and reports NOT_APPLICABLE.

B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

C96005B requires the range of type DURATION to be different from those of its base type; in this implementation there are no values between DURATION'FIRST and DURATION'BASE'FIRST.

CA2009C and CA2009F compile the bodies of generic units separately and following a compilation that contains instantiations of those units. This compiler requires that a generic unit's body be compiled prior to instantiation, and so the unit containing the instantiations is rejected.

BC3204C and BC3204F compile the bodies of generic units separately and following a compilation that contains instantiations of those units. This compiler requires that a generic unit's body be compiled prior to instantiation, and so the unit containing the instantiations is rejected.

AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

CE2105A..B (2 tests), CE2111H, CE2407A, CE3104A, CE3109A this implementation does not allow the creation of a file when FILE_MODE is set to IN_FILE.

CE2107A..C (3 tests) this implementation does not allow more than one internal sequential file to be associated with the same external file.

CE2107D..E (2 tests), CE2107H..I (2 tests), CE2108A, CE2108C, CE3112A this implementation does not allow temporary files to have a name.

CE2107G this implementation does not allow more than one internal direct file with mode OUT_FILE or mode INOUT_FILE to be associated with the same external file.

CE2110B, CE2111D, CE3111A..E (5 tests), CE3114B, CE3115A this implementation does not allow more than one internal text file to be associated with the same external file.

The following 19 tests require a floating-point accuracy that exceeds the maximum of 28 digits supported by this implementation:

C24113Y	(01 test)	C35705Y	(01 test)
C35706Y	(01 test)	C35707Y	(01 test)
C35708Y	(01 test)	C35802Y..Z	(02 tests)
C45241Y	(01 test)	C45321Y	(01 test)
C45421Y	(01 test)	C45521Y..Z	(02 tests)
C45524Y..Z	(02 tests)	C45621Y..Z	(02 tests)
C45641Y	(01 test)	C46012Y..Z	(02 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made with the approval of the AVO, and are made in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into sub-tests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 70 Class B tests (74 test files).

The following Class B test files were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B22003A B26001A B26002A B26005A B28001D B29001A
B2A003A B2A003B B2A003C B33301A B35101A B37106A
B37301B B37302A B38003A B38003B B38009A B38009B
B51001A B53009A B54A01C B54A01J B54A01K B55A01A
B61001C B61001D B61001F B61001H B61001I B61001M
B61001R B61001W B66001C B67001A B67001C B67001D
B91001A B91002A B91002B B91002C B91002D B91002E
B91002F B91002G B91002H B91002I B91002J B91002K
B91002L B95030A B95061A B95061F B95061G B95077A
B97101A B97101E B97102A B97103E B97104G BA1101B0M
BA1101B1 BA1101B2 BA1101B3 BA1101B4 BC1109A BC1109C
BC1109D BC1202A BC1202B BC1202E BC1202F BC1202G
BC2001D BC2001E

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the CYBER 180 Ada Compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the CYBER 180 Ada Compiler using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a CYBER 180-930-31 operating under NOS/VE, Level 700; the host and targe computers were the same.

A magnetic tape containing all tests except for withdrawn tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were not included in their modified form on the magnetic tape. The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the CYBER 180-930-31, and all executable tests were run on the CYBER 180-930-31.

The compiler was tested using command scripts provided by Control Data Corporation and reviewed by the validation team. The compiler was tested using all default (option/switch) settings except for the following:

Option/Switch	Effect
INPUT	Name of the source text file
PROBRAM LIBRARY	Name of the program library
LIST	Name of the source listing file
ERROR	Name of the error file

Tests were compiled, linked, and executed using a single host/target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF.

3.7.3 Test Site

Testing was conducted at Sunnyvale, California and was completed on 24 June 1988.

APPENDIX A

CONFORMANCE STATEMENT

APPENDIX A

CONFORMANCE STATEMENT

Compiler Implementor:

Control Data Corporation
215 Moffett Park Drive
Sunnyvale, CA 94089

Ada Validation Facility:

Software Standards Validation Group
Institute for Computer Science and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, Maryland 20899

Base Configuration

Base Compiler Name: CYBER 180 ADA Compiler: Version 1.1
Host Architecture ISA:
CYBER 180 - 930-31 OS&VER #: NOS/VE LEVEL 700

Target Architecture ISA:
CYBER 180 - 930-31 OS&VER #: NOS/VE LEVEL 700

DECLARATION OF CONFORMANCE

****

Derived Compiler Registration

Derived Compiler Name: CYBER 180 ADA Compiler: Version 1.1

Host Architecture ISA: CYBER 180 Series NOS/VE LEVEL 700
Target Architecture ISA: CYBER 180 Series NOS/VE LEVEL 700

****

Implementor's Declaration

I, the undersigned, representing Control Data Corporation, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that Control Data Corporation is the owner of record of the ///ada language compiler listed above and, as such, is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler listed in this declaration shall be made only in the owner's name:

Control Data Corporation

Richard J. Clifton, Manager Date: 10/5/88
Richard J. Clifton

Ada* is a registered trademark of the United States Government (Ada Joint Program Office).

Owner's Declaration

I, the undersigned, representing Control Data Corporation, take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

Control Data Corporation

Richard J. Clifton, Manager Date: 10/5/88
Richard J. Clifton

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the CYBER 180 Ada Compiler, Version 1.1, are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A).. Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is


     type INTEGER is range -9_223_372_036_854_775_808 ..
                            9_223_372_036_854_775_807;

     type FLOAT is digits 13 range
                       -16#7.FFFF_FFFF_FFF8#E1023..
                        16#7.FFFF_FFFF_FFF8#E1023;

     type LONG_FLOAT is digits 28
                       -16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023..
                        16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023;

     type DURATION is delta 0.001 range
                       -8.589934591999#E09 .. 9.765625000000#E-04;

end STANDARD;
```

Implementation-Dependent Characteristics F

Implementation-Dependent Characteristics F

This appendix summarizes the implementation-dependent characteristics of NOS/VE Ada by listing the following:

- NOS/VE Ada pragmas
- NOS/VE Ada attributes
- Specification of the package SYSTEM
- Restrictions on representation clauses
- Implementation-dependent names
- Address clauses and interrupts
- Unchecked type conversions
- Input-output packages
- Other implementation-dependent characteristics

Shading is not used in this appendix.

F.1 NOS/VE Ada Pragmas

NOS/VE Ada supports the following pragmas as described in the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language, except as shown below:

- INLINE

  This pragma causes inline expansion of a subprogram except as described in annex B of this manual (see 6.3.2, 10.6).

- INTERFACE

  This pragma is supported for CYBIL, FORTRAN, and the NOS/VE Math Library, as discussed in 13.9.1, 13.9.2, and 13.9.3, respectively.

- PACK

  Objects of the given type are packed into the nearest 2**n bits.

- SHARED

  This pragma is not supported for the following types of variables:

  - Variables of type LONG_FLOAT
  - Variables of a subtype of type LONG_FLOAT
  - Variables of a type derived from type LONG_FLOAT
  - Variables of a subtype derived from type LONG_FLOAT

- SUPPRESS

  This pragma is supported, but it is not possible to restrict the check suppression to a specific object or type.

NOS/VE Ada does not support the following pragmas:

- CONTROLLED
- MEMORY_SIZE
- OPTIMIZE
- STORAGE_UNIT
- SYSTEM_NAME

NOS/VE Ada supports the following implementation-defined pragmas:

- COMMON

  This pragma accepts the name of a FORTRAN labeled common as its single argument. This pragma is allowed only in the specification of a library package. This pragma specifies that the library package specification can be accessed as a labeled common by a FORTRAN subroutine. To ensure proper results, the items declared in the Ada library package specification must be of a type corresponding to the type of the matching items in the FORTRAN common specification. The argument name must be a legal NOS/VE and FORTRAN name.

- EXPORT

  This pragma accepts a language name and a subprogram name as arguments. This pragma is allowed only in the body of a library procedure. This pragma specifies the other language (FORTRAN) and informs the Ada compiler that it must provide an entry point in the procedure by the specified subprogram name. (FORTRAN is the only supported language.) The subprogram name must be a NOS/VE and FORTRAN legal name. Parameter passing is not supported.

F.2 NOS/VE Ada Attributes

NOS/VE Ada supports all the attributes described by the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language. It does not provide any implementation-defined attributes. The NOS/VE implementation of the P'ADDRESS attribute returns the prefix P, the 48-bit process virtual address (PVA) right-justified within a 64-bit variable of the predefined type INTEGER.

F.3 Specification of the Package SYSTEM

```
package SYSTEM is
   type ADDRESS is access INTEGER;
   type NAME is (CYBER180);

   SYSTEM_NAME : constant NAME := CYBER180;

   STORAGE_UNIT : constant := 64; -- 64-bit machine
   MEMORY_SIZE : constant := 128*1048576; -- 128 megabytes

   MIN_INT : constant := -9_223_372_036_854_775_808; -- (-2**63)
   MAX_INT : constant := 9_223_372_036_854_775_807; -- (2**63)-1
   MAX_DIGITS : constant := 28;
   MAX_MANTISSA : constant := 63;
   FINE_DELTA : constant := 2#1.0#E-63; -- 2**(-63)
   TICK : constant := 0.001;

   subtype PRIORITY is INTEGER range 0 .. 127;

end SYSTEM;
```

F.4 Restrictions on Representation Clauses

NOS/VE Ada implements representation clauses as described by the ANSI standard for Ada. It does not allow representation clauses for a derived type.

NOS/VE Ada supports the type representation clauses with some restrictions:

- Length clauses
- Enumeration representation clauses
- Record representation clauses

NOS/VE Ada does not support address clauses or interrupts.

F.4.1 Length Clauses

NOS/VE Ada supports the attributes in the length clauses as follows:

- T'SIZE

  Not supported

- T'STORAGE_SIZE (collection size)

  Supported

- T'STORAGE_SIZE (task activation size)

  Supported

- T'SMALL

  Not supported. The compiler always chooses for SMALL the largest power of 2 not greater than the delta in the fixed accuracy definition of the first named subtype T of a fixed point type.

  For example, NOS/VE Ada uses the declaration:

  ```
  type ADA_FIXED is delta 0.05 range 1.00 .. 3.00;
  ```

  to set the ADA_FIXED'SMALL attribute to 0.03125(2^{-5}),
  the largest power of 2 not greater than **delta** 0.05.

F.4.2 Enumeration Representation Clauses

In NOS/VE Ada *enumeration* representation clauses, the internal codes must be in the range of the predefined type INTEGER.

F.4.3 Record Representation Clauses

NOS/VE Ada implements record representation clauses as described by the ANSI language definition. It does not support alignment clauses in record representation clauses.

The component clause of a record representation clause gives the storage place of a component of a record, by providing the following pieces of data:

- The name gives the name of the record component.
- The simple expression following the reserved word AT gives the address in storage units, relative to the beginning of the record, of the storage unit where the component starts.
- The range in the component clause gives the bit positions, relative to that starting storage unit, occupied by the record component.

NOS/VE Ada supports the range for only those record components of discrete types (integer or enumeration) or arrays of discrete elements. The range must specify 1, 2, 4, 8, 16, 32 or 64 bits. Furthermore, if the range of a record component specifies 8 or more bits, then the first bit position of the range must be a multiple of 8 bits. A range can overlap 2 adjacent storage units.

F.5 Implementation-Dependent Names

NOS/VE Ada does not support implementation-dependent names to be used in record representation clauses.

F.6 Address Clauses and Interrupts

NOS/VE Ada does not support address clauses or interrupts.

F.7 Unchecked Type Conversions

NOS/VE Ada allows unchecked conversions when objects of the source and target types have the same size.

F.8 Input-Output Packages

The discussion of NOS/VE Ada implementation of input-output packages includes the following:

- External files and file objects
- Exceptions for input-output errors
- Low level input-output

F.8.1 External Files and File Objects

NOS/VE Ada can process files created by another language processor as long as the data types and file structures are compatible.

NOS/VE Ada supports the following kinds of external files:

- Sequential access files (see 14.1)
- Direct access files (see 14.1)
- Text input-output files (see 14.3)

F.8.2 Exceptions for Input-Output Errors

The ANSI/MIL-STD-1815A-1983 Reference Manual for the Ada Programming Language describes conditions under which input-output exceptions are raised. In addition to these, NOS/VE Ada raises the following exceptions:

- The exception DATA_ERROR is raised when:
  - An attempt is made to read from a direct file a record that has not been defined.
  - A check reveals that the sizes of the records read from a file do not match the sizes of the Ada variables. NOS/VE Ada performs this check except in those few instances where it is too complicated to do so (see 14.2.2).

- The exception USE_ERROR is raised when:
  - The function NAME references a temporary file (see 14.2.1).
  - An attempt is made to delete an external direct file with multiple accesses while more than one instance of open is still active. The file remains open and the position is unchanged (see 14.2.1).
  - An attempt is made to create a sequential, text, or direct file of mode IN_FILE (see 14.2.1).
  - An attempt is made to create an existing file (see 14.2.1).
  - An attempt is made to process a text file with a line that is longer than 511 characters.
  - An attempt is made to set the page length for a text file that does not have the file contents of LIST (see 14.3.3).
  - An attempt is made to issue a new page for a text file that does not have the file contents of LIST.
  - An attempt is made to open or create a file with the FORM parameter specifying anything other than an empty string for sequential access or direct access files.
  - An attempt is made to set a line for a text file that does not have the file contents of LIST and the value specified by TO is less than the current line number.
  - An attempt is made to open or create a text file with the FORM parameter specifying any other value than LIST, LEGIBLE, or UNKNOWN.
  - An attempt is made to open or create a text file with attribute FILE_ CONTENTS not matching the file format specfied by the FORM parameter.

F.8.3 Low Level Input-Output

NOS/VE Ada does not support the package LOW_LEVEL_IO.

F.9 Other Implementation-Dependent Characteristics

The other implementation-dependent characteristics of NOS/VE Ada are discussed as follows:

- Implementation features
- Entity types
- Tasking
- Interface to other languages
- Command interfaces
- Values of data attributes

F.9.1 Implementation Features

The NOS/VE Ada implementation features are listed as follows:

- Predefined types
- Basic types
- Compiler
- Definition of a main program
- TIME type
- Machine code insertions

F.9.1.1 Predefined Types

NOS/VE Ada implements all the predefined types described by the ANSI/MIL-STD-1815A-1983, Reference Manual for the Ada Programming Language, except:

- LONG_INTEGER
- SHORT_FLOAT
- SHORT_INTEGER

F.9.1.2 Basic Types

The sizes of the basic types are as follows:

Type	Size (bytes)
ENUMERATION	8
FLOAT	8
INTEGER	8
LONG_FLOAT	16
TASK	8

In NOS/VE Ada, the enumeration type includes predefined type boolean and character as well as user defined enumeration types.

F.9.1.3 Compiler

NOS/VE Ada provides an ANSI standard Ada compiler.

The NOS/VE Ada compiler has the following characteristics:

- Source code lines up to 132 characters long
- Up to 100 static levels of nesting of blocks and/or subprograms
- External files up to one segment, 2**31-1 bytes, in length
- A generic body can be compiled in a separate file from its specification if it is compiled before it is instantiated. If the specification, body and instantiation are in the same file, the instantiation of the generic can be either before or after the compilation of the body.
- A generic non-library package body or a generic non-library subprogram body cannot be compiled as a subunit in a separate file from its specification.

For Better Performance

The compiler throughput improves when multiple compilation units are submitted. However, if the number of compilation units grows over a certain limit, for example 50 small compilation units of about 50 lines each, or if the first compilation units are large, the throughput actually degrades.

Using the pragma INLINE, where applicable, results in faster object code by avoiding the call/return instructions.

F.9.1.4 Definition of a Main Program

NOS/VE Ada requires that the main program be a procedure without parameters. The name of a compilation unit used as a main program must follow NOS/VE naming standards. The name can be up to 31 characters in length and must be a valid NOS/VE name and a valid Ada identifier. Any naming error is detected at link time only. For more information, see the Ada for NOS/VE Usage manual.

F.9.1.5 TIME Type

NOS/VE Ada defines the type TIME as an integer representing the Julian date in milliseconds.

F.9.1.6 Machine Code Insertions

NOS/VE Ada does not support machine code insertions.

F.9.2 Entity Types

This discussion contains information on:

- Array types
- Record types
- Access types

F.9.2.1 Array Types

Arrays are stored row wise, that is, the last index changes the fastest.

An array has a type descriptor that NOS/VE Ada uses when the array is one of the following:

- A component of a record with discriminants
- Passed as a parameter
- Created by an allocator

For each index, NOS/VE Ada builds the following information triplet:

Lower Bound
Upper Bound
Element Size

For multi-dimension arrays, NOS/VE Ada allocates the triplets consecutively.

Element size is expressed in number of storage units (64-bit words). If the array is packed, the element size is expressed in number of bits and represented by a negative value.

NOS/VE Ada strings are packed arrays of characters. Each component of the array is an 8-bit (1-byte) character. Packed arrays of booleans use 1 bit per component and are left-justified. Arrays of integers or enumeration variables can also be packed. Each component uses n bits. Thus, the integer or enumeration subtype is in the range -2**n .. (2**n)-1.

Note that all objects start on a storage unit (64-bit word) boundary.

At run time when NOS/VE Ada elaborates an array definition, the amount of available space remaining either on the stack or in the heap limits the maximum size of the array (see 3.6).

F.9.2.2 Record Types

At run time when NOS/VE Ada elaborates a record definition, the amount of available space remaining either on the stack or in the heap limits the maximum size of the record (see 3.7).

NOS/VE Ada raises the exception STORAGE_ERROR at run time when the size of an elaborated object exceeds the amount of available space.

The rest of this discussion on how records are stored includes:

- Simpie record types (without discriminants)
- Record types with discriminants

F.9.2.2.1 Simple Record Types (Without Discriminants)

In the absence of representation clauses, each record component is word aligned. NOS/VE Ada stores the record components in the order they are declared.

A fixed size array (lower and upper bounds are constants) is stored within the record. Otherwise, the array is stored elsewhere in the heap, and is replaced by a pointer to the array value (first element of the array) in the record.

F.9.2.2.2 Record Types With Discriminants

The discriminants are stored first, followed by all the other components as described for simple records.

If a record component is an array with index values that depend on the value of the discriminant(s), the array and its descriptor are both allocated on the heap. They are replaced by a pair of pointers in the record. One points to the array value and the other points to the array descriptor.

F.9.2.3 Access Types

Objects of access type are 6-byte pointers, left-justified within a word, to the accessed data contained in some allocated area in the heap. If the accessed data is of type array or packed array, the allocated area also contains the address of the array descriptor in front of the data.

F.9.3 Tasking

NOS/VE Ada supports tasking by running all Ada tasks as NOS/VE concurrent procedures activated and controlled by the tasking kernel which is an integral part of the NOS/VE compiler run time system. Contact the site administrator to change the site's TASK_LIMIT to run more concurrent tasks than the site currently allows. See the Ada for NOS/VE Usage manual for more information on NOS/VE Ada tasking.

F.9.4 Interfaces to Other Languages

NOS/VE Ada supports calls to CYBIL and FORTRAN subprograms and to NOS/VE Math Library subroutines with the following restrictions:

- CYBIL interface

  (See 13.9.1 and chapter 6 of the Ada for NOS/VE Usage manual).

- FORTRAN interface

  (See 13.9.2 and chapter 6 of the Ada for NOS/VE Usage manual).

- Math Library interface

  (See 13.9.3 and chapter 6 of the Ada for NOS/VE Usage manual).

F.9.5 Command Interfaces

The discussion of the command interfaces implemented by NOS/VE Ada includes:

- Program Library Utility commands
- Compiler command
- Linker command
- Execution commands

NOS/VE Ada commands use the syntax and language elements for parameters described in the NOS/VE System Usage manual.

F.9.5.1 Program Library Utility Commands

NOS/VE Ada provides a hierarchically structured (tree structured) program library to fulfill the ANSI Ada language definition requirements. A node (sublibrary) in the tree can contain up to 4096 compilation units. The Ada for NOS/VE Usage manual contains a detailed discussion of the NOS/VE Ada implementation of the program library.

F.9.5.2 Compiler Command

The NOS/VE Ada compiler can compile an ANSI standard Ada program on NOS/VE. See the Ada for NOS/VE Usage manual for information about the NOS/VE Ada compiler command.

F.9.5.3 Linker Command

The NOS/VE Ada linker checks the order of compilation of the compilation units of a program before the program can be executed.

See the Ada for NOS/VE Usage manual for more information about the linker command.

F.9.5.4 Execution

NOS/VE Ada provides several ways to load and execute an Ada program. They are described in the following manuals:

- Ada for NOS/VE Usage
- CYBIL for NOS/VE System Interface Usage
- NOS/VE Object Code Management Usage

F.9.6 Values of Data Attributes

The package STANDARD contains the declaration of the following predefined types and their attributes:

- Integer (INTEGER)
- Floating point (FLOAT)
- Long floating point (LONG_FLOAT)
- Duration (DURATION)

F.9.6.1 Values of Integer Attributes

Attribute	Value
FIRST	-9_223_372_036_854_775_808
LAST	9_223_372_036_854_775_807
SIZE	64
WIDTH	20

F.9.6.2 Values of Floating Point Attributes

Attribute	Value
DIGITS	13
EMAX	180
EPSILON	5.6_843_419_961#E-14
FIRST	-16#7.FFFF_FFFF_FFF8#E1023
LARGE	1.532495540866E54
LAST	16#7.FFFF_FFFF_FFF8#E1023
MACHINE_EMAX	4095
MACHINE_EMIN	-4096
MACHINE_MANTISSA	48
MACHINE_OVERFLOWS	TRUE
MACHINE_RADIX	2
MACHINE_ROUNDS	FALSE
MANTISSA	45
SAFE_EMAX	4095
SAFE_LARGE	5.221944407066E1232
SAFE_SMALL	9.574977460952E-1234
SIZE	64
SMALL	3.262652233999E-55

F.9.6.3 Values of Long Floating Point Attributes

Attribute	Value
DIGITS	28
EMAX	380
FIRST	-16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023
LARGE	2.4626253872746549507674400006E114
LAST	16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023
MACHINE_EMAX	4095
MACHINE_EMIN	-4096
MACHINE_MANTISSA	96
MACHINE_OVERFLOWS	TRUE
MACHINE_RADIX	2
MACHINE_ROUNDS	FALSE
MANTISSA	95
SAFE_EMAX	4095
SAFE_LARGE	5.2219444070657625334587635552E1232
SAFE_SMALL	9.5749774609521853579467310011E-1234
SIZE	128
SMALL	2.0303534698525193786192196645E-115

F.9.6.4 Values of Duration Attributes

Attribute	Value
DELTA	1.000000000000E-03
LARGE	8.589934591999E09
MACHINE_OVERFLOWS	TRUE
MACHINE_ROUNDS	FALSE
SIZE	64
SMALL	9.765625000000E-04

NOS/VE Ada Packages 4

A package specifies a group of logically related entities, such as types, objects of those types, and subprograms with parameters of those types. Ada includes three classes of packages:

- Predefined package STANDARD
- Predefined library packages such as SYSTEM, CALENDAR, and TEXT_IO
- User-defined packages

Packages enable you to express and enforce an abstract solution in the syntax of Ada. The package structure enables system designers to modularize a solution. The separation of package specifications enables the designers to identify necessary packages, and then develop the bodies later. This chapter presents an overview of package structure and describes the NOS/VE Ada implementation-defined packages.

Package Structure

Packages provide a convenient structure for grouping logically related items (for example, groups of common declarations or groups of subprograms). Packages, like other Ada program units, have a two-part structure: a specification and a body.

Package Specifications

During the design phase of your project, you can code your package specifications as part of your high-level design. You can also decide what information you want to make visible to your users and what information you prefer to hide from them. Once you reach the development phase of your project, your development team can code the package bodies. Through the use of stubs and separate compilation, as described in the Ada for NOS/VE Reference Manual, a system engineer can specify the necessary types, objects, and subprograms for later development.

Declarations create instances of a given type. Through object declarations, you can create variables and constants.

Package Bodies

The information hiding feature within Ada enables you to code a body of a package and then store it as a private segment of code.

Implementation-Defined Packages

NOS/VE Ada Implementation Feature

Once you create your NOS/VE Ada program library, as described in chapter 2, you gain immediate access to the following system supplied packages:

Package	Defines
STANDARD	Predefined number types
SYSTEM	Hardware characteristics
CALENDAR	Clock and calendar
IO_EXCEPTIONS	Run time exceptions
TEXT_IO	Input-output for string, character, integer, real, and .numeration types
SEQUENTIAL_IO	Sequential file management
DIRECT_IO	Direct file management

By using the SHOW command from inside the Ada Program Library Utility, you can display the status of each package, including the date of the last compilation of each specification and body.

Package STANDARD

The package STANDARD contains the declarations of the predefined types: INTEGER, FLOAT, LONG_FLOAT, and DURATION.

Type FLOAT provides single precision accuracy of 13 significant decimal digits. Type LONG_FLOAT provides double precision accuracy of 28 significant decimal digits.

All numbers in package STANDARD are decimal unless otherwise indicated. Figure 4-1 displays the notation conversion used for the values listed in this package. Note that the exponent indicates the power of the base by which the preceding number must be multiplied. For example, 4#2#E3 means $2*(4^3)$ or 128; the base is 4.

```
with TEXT_IO;
use TEXT_IO;

procedure NOTATION_EXAMPLE is
package INTIO is new INTEGER_IO (INTEGER);
use INTIO;
x : INTEGER := 4#2#E3;

begin
   PUT ("4#2#E3 equals: ");
   PUT(x);
   NEW_LINE;
   PUT_LINE ("In arithmetic notation that means 2*(4 cubed).");
end NOTATION_EXAMPLE;
```

Figure 4-1. Procedure NOTATION_EXAMPLE

Figure 4-2 lists the INTEGER, FLOAT, LONG_FLOAT, and DURATION types and their attributes as provided by package STANDARD.

```
type INTEGER
   INTEGER'FIRST := -9_223_372_036_854_775_808;
   INTEGER'LAST := 9_223_372_036_854_775_807;
   INTEGER'SIZE := 64;

type FLOAT
   FLOAT'DIGITS := 13; – Single Precision
   FLOAT'EMAX := 180;
   FLOAT'EPSILON := 5.6_843_419_961#E-14;
   FLOAT'FIRST := -16#7.FFFF_FFFF_FFF8#E1023;
   FLOAT'LARGE := 1.532495540866#E54;
   FLOAT'LAST := 16#7.FFFF_FFFF_FFF8#E1023;
   FLOAT'MACHINE_EMAX := 4095;
   FLOAT'MACHINE_EMIN := -4096;
   FLOAT'MACHINE_MANTISSA := 48;
   FLOAT'MACHINE_OVERFLOWS := TRUE;
   FLOAT'MACHINE_RADIX := 2;
   FLOAT'MACHINE_ROUNDS := FALSE;
   FLOAT'MANTISSA := 45;
   FLOAT'SAFE_EMAX := 4095;
   FLOAT'SAFE_LARGE := 5.221944407066#E1232;
   FLOAT'SAFE_SMALL := 9.574977460952#E-1234;
   FLOAT'SMALL := 3.26265223999#E-55;
   FLOAT'SIZE := 64;

type LONG_FLOAT
   LONG_FLOAT'DIGITS := 28; – Double Precision
   LONG_FLOAT'EMAX := 380;
   LONG_FLOAT'FIRST :=
      -16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023;
   LONG_FLOAT'LARGE :=
      2.462625387274654950767440006#E114;
   LONG_FLOAT'LAST :=
      16#7.FFFF_FFFF_FFFF_FFFF_FFFF_FFF8#E1023;
   LONG_FLOAT'SMALL :=
      2.030353469852519378619219645#E-115;
   LONG_FLOAT'MACHINE_EMAX := 4095;
   LONG_FLOAT'MACHINE_EMIN := -4096;
   LONG_FLOAT'MACHINE_MANTISSA := 96;
   LONG_FLOAT'MACHINE_OVERFLOWS := TRUE;
   LONG_FLOAT'MACHINE_RADIX := 2;
   LONG_FLOAT'MACHINE_ROUNDS := FALSE;
   LONG_FLOAT'MANTISSA := 95;
   LONG_FLOAT'SAFE_EMAX := 4095;
   LONG_FLOAT'SAFE_LARGE :=
      5.221944407065762533458763552#E1232;
   LONG_FLOAT'SAFE_SMALL := 9.574977460952185357946731011#E-1234;
   LONG_FLOAT'SIZE := 128;
```

Figure 4-2. Package STANDARD Excerpts

(Continued)

(Continued)

```
type DURATION
   DURATION'DELTA := 1.000000000000#E-03;
   DURATION'LARGE := 8.589934591999#E09;
   DURATION'MACHINE_ROUNDS := FALSE;
   DURATION'MACHINE_OVERFLOWS := TRUE;
   DURATION'SMALL := 9.765625000000#E-04;
   DURATION'SIZE := 64;
```

Figure 4-2. Package STANDARD Excerpts

Package SYSTEM

The package SYSTEM defines the hardware characteristics of NOS/VE Ada. Figure 4-2 lists package SYSTEM.

```
package SYSTEM is
   type ADDRESS is access INTEGER;
   type NAME is (CYBER180);

   SYSTEM_NAME : constant NAME := CYBER180;
   STORAGE_UNIT : constant NAME := 64;
   – 64-bit machine
   MEMORY_SIZE : constant := 128*1048576;
   – 128 megabytes
   MIN_INT : constant := -9_223_372_036_854_775_808;
   – (-2**63)
   MAX_INT : constant := 9_223_372_036_854_775_807;
   – 2**63-1
   MAX_DIGITS : constant := 28;
   MAX_MANTISSA : constant := 63;
   FINE_DELTA : constant := 2#1.0#E-63;
   – 2**(-63)
   TICK : constant := 0.001;

   subtype PRIORITY is INTEGER range 0..127;

end SYSTEM;
```

Figure 4-3. Package SYSTEM

Table 4-1 summarizes NOS/VE Ada characteristics. Most of the characteristics listed in table 4-1 are from package SYSTEM, but others are defined either by NOS/VE or elsewhere in the NOS/VE Ada compiler.

Table 4-1. NOS/VE Ada Characteristics

Characteristic	Value	Defined By
Minimum integer	-2**63	Package SYSTEM
Maximum integer	2**63-1	Package SYSTEM
Maximum digits	28	Package SYSTEM
Maximum mantissa	63	Package SYSTEM
Maximum line length	132 characters	NOS/VE
Maximum name length for a main program	31 characters	NOS/VE

Package CALENDAR

The package CALENDAR defines the clock and calendar definitions of NOS/VE Ada. Figure 4-3 lists package CALENDAR.

```
package CALENDAR is

   type TIME is private;

   subtype YEAR_NUMBER is INTEGER range 1901 .. 2099;
   subtype MONTH_NUMBER is INTEGER range 1 .. 12;
   subtype DAY_NUMBER is INTEGER range 1 .. 31;
   subtype DAY_DURATION is DURATION range 0.0 .. 86_400.0;

   function CLOCK return TIME;

   function YEAR (DATE : TIME) return YEAR_NUMBER;
   function MONTH (DATE : TIME) return MONTH_NUMBER;
   function DAY (DATE : TIME) return DAY_NUMBER;
   function SECONDS (DATE : TIME) return DAY_DURATION;

   procedure SPLIT (DATE : in TIME;
      YEAR : out YEAR_NUMBER;
      MONTH : out MONTH_NUMBER;
      DAY : out DAY_NUMBER;
      SECONDS : out DAY_DURATION);

   function TIME_OF (YEAR : YEAR_NUMBER;
      MONTH : MONTH_NUMBER;
      DAY : DAY_NUMBER;
      SECONDS : DAY_DURATION := 0.0) return TIME;

   function "+" (LEFT : TIME; RIGHT : DURATION) return TIME;
   function "+" (LEFT : DURATION; RIGHT : TIME) return TIME;
   function "-" (LEFT : TIME; RIGHT : DURATION) return TIME;
   function "-" (LEFT : TIME; RIGHT : TIME) return DURATION;

   function "<" (LEFT, RIGHT : TIME) return BOOLEAN;
   function "<=" (LEFT, RIGHT : TIME) return BOOLEAN;
   function ">" (LEFT, RIGHT : TIME) return BOOLEAN;
   function ">=" (LEFT, RIGHT : TIME) return BOOLEAN;

   TIME_ERROR : exception; – can be raised by TIME_OF, "+", and "-"

private
   – implementation-dependent
end;
```

Figure 4-4. Package CALENDAR

See appendix I for listings of NOS/VE Ada packages TEXT_IO, SEQUENTIAL_IO, and DIRECT_IO.

End of NOS/VE Ada Implementation Feature

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	<131 x "A">1
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	<131 x "A">2
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	<65 x "A">3<66 x"A">
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	<65 x "A">4<66 x"A">
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	<129 x "0">298
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	<126 x "0">69.0E1

$BIG_STRING1 <66 x "A">

A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.

$BIG_STRING2 <65 x "A">1

A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.

$BLANKS <112 x " ">

A sequence of blanks twenty characters less than the size of the maximum line length.

$COUNT_LAST 9223372036854775807

A universal integer literal whose value is TEXT_IO.COUNT'LAST.

$FIELD_LAST 67

A universal integer literal whose value is TEXT_IO.FIELD'LAST.

$FILE_NAME_WITH_BAD_CHARS BAD_CHARS^#.%!X

An external file name that either contains invalid characters or is too long.

$FILE_NAME_WITH_WILD_CARD_CHAR This_File_Name_Has_To_Be_Too_Long_Wild_Card_Char_Do_Not_Exist

An external file name that either contains a wild card character or is too long.

$GREATER_THAN_DURATION 100000000.0

A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.

$GREATER_THAN_DURATION_BASE_LAST 7000000000.0

A universal real literal that is greater than DURATION'BASE'LAST.

$ILLEGAL_EXTERNAL_FILE_NAME1 BADCHARS^@.~!

An external file name which contains invalid characters.

$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	MUCH_TOO_LONG_NAME_FOR_A_VE_ FILE
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-9_223_372_036_854_775_808
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	9_223_372_036_854_775_807
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	9223372036854775808
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100000000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-7000000000.0
$MAX_DIGITS Maximum digits supported for floating-point types.	28
$MAX_IN_LEN Maximum input line length permitted by the implementation.	132
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	9223372036854775807
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	9223372036854775808
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	2:<127 X "0">11:

$MAX_LEN_REAL_BASED_LITERAL 16:<125 X "0">F.E:

A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.

$MAX_STRING_LITERAL "<130 X "A">"

A string literal of size MAX_IN_LEN, including the quote characters.

$MIN_INT -9223372036854775808

A universal integer literal whose value is SYSTEM.MIN_ INT.

$NAME DOES_NOT_EXIST

A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.

$NEG_BASED_INT 8#1<20 x "7">6

A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 27 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

B28003A: A basic declaration (line 36) wrongly follows a later declaration.

E28005C: This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ARG.

C34004A: The expression in line 168 wrongly yields a value outside of the range of the target type T, raising CONSTRAINT_ERROR.

C35502P: Equality operators in lines 62 & 69 should be inequality operators.

A35902C: Line 17's assignment of the nomimal upper bound of a fixed-point type to an object of that type raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.

C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.

C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may in fact raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.

C35A03E,
& R: These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.

C37213H: The subtype declaration of SCONS in line 100 is wrongly expected to raise an exception when elaborated.

C37213J: The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.

C37215C, E, G, H: Various discriminant constraints are wrongly expected to be incompatible with type CONS.

C38102C: The fixed-point conversion on line 23 wrongly raises CONSTRAINT_ERROR.

C41402A: 'STORAGE_SIZE is wrongly applied to an object of an access type.

C45332A: The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOWS is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOWS may still be TRUE.

C45614C: REPORT.IDENT_INT has an argument of the wrong type (LONG_INTEGER).

A74106C, C85018B, C87B04B, CC1311B: A bound specified in a fixed-point subtype declaration lies outside of that calculated for the base type, raising CONSTRAINT_ERROR. Errors of this sort occur re lines 37 & 59, 142 & 143, 16 & 48, and 252 & 253 of the four tests, respectively (and possibly elsewhere).

BC3105A: Lines 159..168 are wrongly expected to be illegal; they are legal.

AD1A01A: The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.

CE2401H: The record aggregates in lines 105 & 117 contain the wrong values.

CE3208A: This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.